Balancing Contention and Synchronization on the Intel Paragon

Shahid H. Bokhari
Department of Electrical Engineering
University of Engineering & Technology
Lahore, Pakistan

David M. Nicol
Department of Computer Science
Dartmouth College
Hanover, New Hampshire

Abstract

The Intel Paragon is a mesh-connected distributed memory parallel computer. It uses an oblivious and deterministic message routing algorithm: this permits us to develop highly optimized schedules for frequently needed communication patterns.

The complete exchange is one such pattern. Several approaches are available for carrying it out on the mesh. We study an algorithm developed by Scott. This algorithm assumes that a communication link can carry one message at a time and that a node can only transmit one message at a time. It requires global synchronization to enforce a schedule of transmissions. Unfortunately global synchronization has substantial overhead on the Paragon. At the same time the powerful interconnection mechanism of this machine permits 2 or 3 messages to share a communication link with minor overhead. It can also overlap multiple message transmission from the same node to some extent.

We develop a generalization of Scott's algorithm that executes complete exchange with a prescribed contention. Schedules that incur greater contention require fewer synchronization steps. This permits us to tradeoff contention against synchronization overhead.

We describe the performance of this algorithm and compare it with Scott's original algorithm as well as with a naive algorithm that does not take interconnection structure into account.

The Bounded contention algorithm is always better than Scott's algorithm and outperforms the naive algorithm for all but the smallest message sizes. The naive algorithm fails to work on meshes larger than 12×12. These results show that due consideration of processor interconnect and machine performance parameters is necessary to obtain peak performance from the Paragon and its successor mesh machines.

Research supported by NASA Contract NAS1-19480, while the authors were in residence at the Institute for Computer Applications in Science & Engineering, NASA Langley Research Center, Hampton, Virginia.

This research was performed using the Trex 512 node Paragon operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by NASA.

Shahid H. Bokhari was additionally supported by a grant from the Directorate of Research Extension and Advisory Services, University of Engineering & Technology, Lahore.

1 Introduction

Interprocessor communication overhead is a major factor that limits the performance of distributed memory parallel computer systems. All machines, no matter how powerful their interprocessor communication mechanism, suffer from this overhead. Communication overhead is exacerbated by *node* and *link contention*. Node contention arises when a node attempts to transmit or receive several messages simultaneously. Link contention is caused by the sharing of a communication link by two or more messages. Contention arises in all but the simplest communication requirements. In some cases, contention can be minimized or eliminated by careful scheduling of messages. However this requires that all processors in the system synchronize themselves at specific points in time—thereby incurring *synchronization overhead*.

The parallel algorithm designer is thus faced with the following dilemma:

- A completely contention-free schedule will incur substantial synchronization overhead.
- A completely synchronization-free schedule will result in heavy contention overhead.

Clearly there is a need to find a balance between the two types of overhead in order to minimize the overall execution time of the parallel algorithm.

The *complete exchange* is an interprocessor communication pattern that arises in a number of important applications. It requires each processor to send a distinct message to every other processor in the system and is thus the heaviest communication requirement that can be imposed on a parallel computer. Complete exchange has been extensively studied and a number of algorithms are known for its efficient execution on various interconnection networks.

We describe a study of the complete exchange on mesh connected parallel machines. We start with an algorithm to execute the complete exchange on meshes that was developed by David Scott. We develop a generalization of this algorithm that permits us to decrease synchronization overhead by increasing contention. We describe our experiments with this approach on the 512-node Intel Paragon mesh at Caltech. It is seen that the generalized algorithm can be used to balance contention and synchronization overhead and thus obtain significant reduction in the time required to execute the complete exchange. The generalized algorithm is also shown to give better performance than a naive algorithm that does not take the interconnect of the Paragon into account.

Our results demonstrate that careful consideration of parallel machine interconnect and performance characteristics is needed in order to obtain the best performance. As an extreme example, the naive algorithm (which does not take the interconnect into account) fails to execute on Paragon meshes of size larger than 12×12, because the operating system cannot allocate enough memory for the large amount of communication traffic required. For such meshes we have no choice but to use an

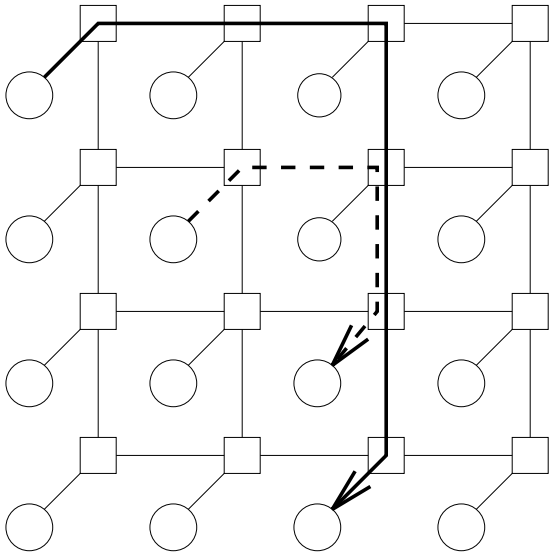

Figure 1: The mesh interconnect of a 4×4 Paragon. The circles represent compute nodes while the squares show special purpose hardware for communication. Message routing is done via the "row–column" algorithm explained in the text. The figure shows two pairs of processors communicating and contending for a single edge. Such *link contention* can lead to substantial overhead.

algorithm that carefully schedules communications, such as Scott's algorithm or its generalization (described in this paper).

2 The Paragon Mesh

The mesh has long been a popular choice for interconnecting parallel computers. Currently, the most powerful example of the mesh is the Intel Paragon[1]. The specific machine on which the experiments described in this paper were carried out is located at the Center for Advanced Computing Research at Caltech[2]. It is made up of 512 compute nodes organized in a 16×32 array. Each node is composed of two Intel i860 processors. One serves as a compute processor and the other as a communication processor. In addition there is special hardware for interfacing with the intercommunication network. The interprocessor communication network is a mesh with "row–column" routing (Figure 1). A message traveling from source s to destination t first travels along the row in which s lies, until it reaches the column in which t lies; it then travels along the column to t. Two messages traveling simultaneously between two different source-destination pairs may need to traverse the same communication link, as illustrated in Figure 1, and will incur link contention

[1] `http://www.ssd.intel.com/paragon.html`

[2] `http://www.cacr.caltech.edu`

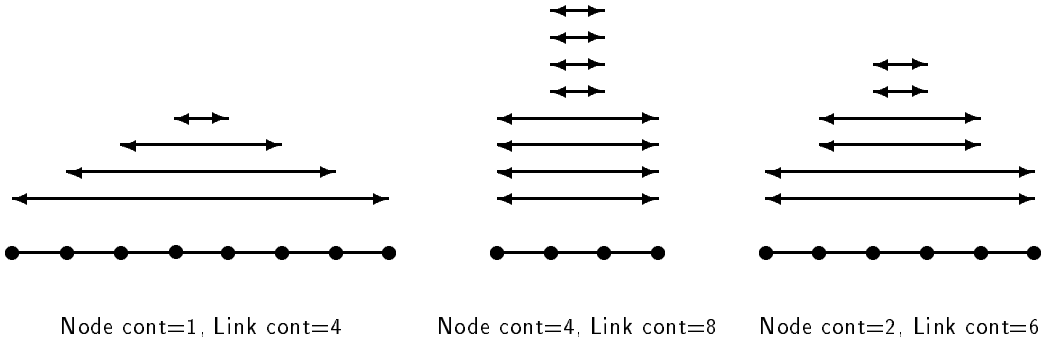

Figure 2: Explanation of node and link contention on chains of processors. Node contention equals the number of messages that a processor attempts to transmit simultaneously. Link contention is given by the maximum number of messages passing through any communication link in the chain.

overhead.

The routing mechanism on the Paragon is *oblivious* (the paths between all source-destination pairs are statically defined) and *deterministic* (a single route exists between every source-destination pair). As a result, it is possible to accurately predict the time required for a communication step, provided no contention is taking place.

A message passing through a node *en route* to its destination does not impact the computation occuring at that node as the routing is carried out by special hardware. The i860s run at 50 MHz and are capable of 75 MFlops. This machine has 32 Megabytes of memory per node of which about 24 Megabytes are available for user programs. Measured performance parameters of the Paragon are given in Table 1. The communication expression in this table is obtained by using the specific communication scheme employed in subsequent experiments with the complete exchange and thus differs from the expressions reported elsewhere [2, 5].

Table 1: Performance Parameters for the Paragon

Synchronization $n \times n$ processors	$274 \log_2 n - 134$ μsec
Communication, message $m \geq 8640$ bytes	$231 + 0.022m$ μsec

Figure 2 clarifies the concepts of node and link contention, as applied to chains of processors. The interpretation of these concepts for meshes is very similar though difficult to explain in a simple diagram.

The successor machine to the Paragon is the Intel ASCI (Accelerated Strategic Computing Initiative) Teraflop[3][12], which is currently being installed at Sandia Laboratories[4]. This machine also has a mesh interconnect and the techniques de-

[3] `http://www.ssd.intel.com/tflop.html`

[4] `http://www.cs.sandia.gov/teraflop.html`

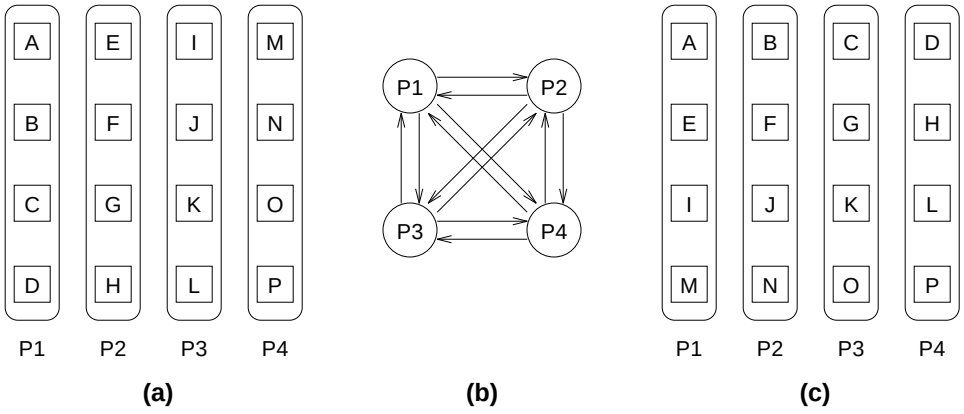

Figure 3: Complete Exchange on 4 Processors. To change storage from column order (a) to row order (c), each processor must send a distinct message to every other processor (b).

scribed in this paper should be applicable to the new machine as well.

3 The Complete Exchange

On a distributed memory parallel computer, the *complete exchange* requires each of N processors to send a distinct m byte block to each of the remaining $N-1$ processors. This communication pattern, which is also known as *all-to-all personalized,* is at the heart of many important multicomputer algorithms such as matrix transposition, matrix-vector multiply, Fast Fourier Transforms and the Alternating Directions Implicit (ADI) method for solving partial differential equations. To understand the data movement required by this pattern refer to Figure 3 which shows a 4×4 block matrix stored on 4 processors. In part (a) of this Figure the matrix is stored in column order. In part (c) the layout has been changed to row order. It is clear that to change from (a) to (c), each processor must transmit a block of data to every other processor. This is shown in part (b) which is a complete directed graph of four nodes.

In general, complete exchange on N processors can be represented by a complete directed graph of N nodes. It is thus the densest possible communication requirement and the time required by a distributed memory multicomputer to execute it is an important performance parameter. At the same time, it is a challenge for the algorithm designer to develop good algorithms for complete exchange on different parallel architectures.

A number of algorithms have been developed for executing the complete exchange on hypercubes [4, 6, 7, 8, 11] and meshes [1, 9]. These algorithms attempt to obtain high performance by carefully scheduling communications so as to avoid node and link contention. We can classify these algorithms into two categories. In *Direct* algorithms each block is transmitted once to its ultimate destination; in *Store-and-*

forward algorithms a block is combined with others and transmitted in stages via intermediate processors. Store-and-forward algorithms [7] strive to reduce the impact of startup time by incurring data permutation and extra transmission overhead. It has been shown that such algorithms perform well for small message sizes. Direct algorithms [11, 9], on the other hand, have better performance for large message sizes.

The time required to execute the complete exchange will depend on the interconnection network and the schedule of data transfers. We shall address the problem of developing good direct algorithms for mesh connected parallel architectures. The sparsity of the mesh interconnect makes this a difficult endeavor. This is in contrast with hypercubes, for which optimal direct algorithms (i.e., those that require $N-1$ transmissions for an N processor system) have been known for some time.

4 Scott's Algorithm

The problem of implementing complete exchange on a mesh architecture has been studied by Scott [9] under the following assumptions:

- A node can send and receive at most one message at a time.
- A communication link can carry at most one message in each direction at one time.
- Messages are routed according to the "row-column" algorithm, that is, a message from processor x_1, y_1 to processor x_2, y_2 first travels along a row to x_2, y_1 and then along a column to x_2, y_2.

Scott shows that, under these assumptions, a square mesh of N nodes cannot achieve the complete exchange in fewer than $N^{3/2}/4$ steps, unlike a hypercube, which requires $N-1$ steps. The intuitive reason for this is the far richer interconnection of the hypercube which comes, of course, at the cost of a logarithmically increasing node degree.

Scott goes on to describe a procedure that will generate a schedule of transmissions that takes exactly $N^{3/2}/4$ steps, for the case where N is a multiple of 4. This procedure is based on composing or "cross-multiplying" pairs of 1-dimensional permutations and can lead to many different sets of schedules, depending on the choices made when composing the permutations. Figure 4 shows three permutations out of a set of 128 generated for an 8×8 mesh. The cells in this diagram are assumed to be numbered in row-major order. A non-blank cell indicates the coordinates of the target to which the corresponding processor has to transmit. A blank cell indicates that the corresponding processor *does not* transmit anything during that permutation. As we increase the size of the mesh, the proportion of these idle processors increases because the mesh interconnect cannot support transmissions by all processors. It is these idle processors that lead to the superlinear $N^{3/2}/4$ expression for run time.

		1,4	1,2	1,5	1,3		
		7,4	7,2	7,5	7,3		
3,6	3,0					3,7	3,1
5,6	5,0					5,7	5,1
2,6	2,0					2,7	2,1
4,6	4,0					4,7	4,1
		0,4	0,2	0,5	0,3		
		6,4	6,2	6,5	6,3		

2,1	2,7					2,0	2,6
		4,3	4,5	4,2	4,4		
7,1	7,7					7,0	7,6
		1,3	1,5	1,2	1,4		
		6,3	6,5	6,2	6,4		
0,1	0,7					0,0	0,6
		3,3	3,5	3,2	3,4		
5,1	5,7					5,0	5,6

3,5		3,0			3,7		3,2
	2,3		2,6	2,1		2,4	
	6,3		6,6	6,1		6,4	
7,5		7,0			7,7		7,2
0,5		0,0			0,7		0,2
	1,3		1,6	1,1		1,4	
	5,3		5,6	5,1		5,4	
4,5		4,0			4,7		4,2

Figure 4: Three out of a set of 128 permutations for an 8×8 mesh. The cells in this figure represent processors and are numbered in row-major order. An empty cell indicates an inactive processor. A non-empty cell gives the coordinates of the cell to which that cell transmits.

5 Bounded Contention Algorithm

The permutations generated by Scott's procedure assume that only one message can travel over a link in one direction at a time. As a result all nodes cannot, in general, transmit during any given step. This is evident in Figure 4, where we see that half the nodes are always inactive. If we have a square mesh of $n \times n = N$ nodes, the number of steps required is $n^3/4 = N^{3/2}/4$ and during each step a fraction $4/n$ of the nodes is inactive.

If we relax the constraint that a link only carry one message at a time, it becomes interesting to explore if schedules can be generated in which contention is bounded by some integer c. The permutations shown in Figure 4 cannot simply be superimposed because the active nodes in any pair of permutations are not disjoint.

Scott's generation technique creates permutations that can be executed in any order to achieve the complete exchange. The set of permutations generated is not unique. We have developed an algorithm to generate a set of permutations in a special *collapsible* order. This generates permutations in such a way that consecutive entries in the sequence can be collapsed to form a denser permutation (i.e., one in which more nodes are active), with greater contention. The collapsibility property is not true of Scott's permutations in general.

Figure 5 shows two permutations for an 8×8 mesh that can be collapsed to form a third. Since each of the constituent permutations has link contention bounded by 1, the contention in the collapsed permutation is bounded by 2. It is also clear that each node is transmitting exactly once.

For the 8×8 mesh shown in Figure 5, the fraction of active nodes in the constituent permutations is $4/n = 1/2$. We can combine sets of two permutations each and thus halve the number of steps required to achieve complete exchange.

We have developed a theory of collapsible schedules for the complete exchange on meshes. We can show that for a square mesh of $n \times n = N$ nodes that permits contention c on its links, the number of steps required is $n^3/4c$, where c is an integer $\leq n/4$ and c divides $n/4$ (i.e., $n/4c$ is an integer).

We have implemented an algorithm based on this theory and used it to generate and verify schedules for meshes of size 4×4, 8×8, ..., 32×32[5]. Table 2 shows the improvement possible as the permitted contention is allowed to increase. For each mesh size, the minimum steps possible are n^2 at $c = n/4$. This is within 1 of the theoretical minimum $n^2 - 1$. The blank entries below the principal diagonal in Table 2 are caused by the constraint that $n/4c$ be an integer. This table assumes that no node contention is permitted, i.e., a node cannot transmit more than one message at a time.

The schedules generated by this algorithm have the interesting property that they can be collapsed to whatever degree is permitted by the rules stated above. Thus the schedule for 16×16 meshes could be collapsed for link contention 2 or 4 by combining

[5]Schedules for meshes of size 4×4, 8×8, 12×12 and 16×16 are available at the following site: `ftp://ftp.icase.edu/pub/cs/shahid`

1,1	1,7					1,0	1,6
7,1	7,7					7,0	7,6
		3,3	3,5	3,2	3,4		
		5,3	5,5	5,2	5,4		
		2,3	2,5	2,2	2,4		
		4,3	4,5	4,2	4,4		
0,1	0,7					0,0	0,6
6,1	6,7					6,0	6,6

		1,3	1,5	1,2	1,4		
		7,3	7,5	7,2	7,4		
3,1	3,7					3,0	3,6
5,1	5,7					5,0	5,6
2,1	2,7					2,0	2,6
4,1	4,7					4,0	4,6
		0,3	0,5	0,2	0,4		
		6,3	6,5	6,2	6,4		

1,1	1,7	1,3	1,5	1,2	1,4	1,0	1,6
7,1	7,7	7,3	7,5	7,2	7,4	7,0	7,6
3,1	3,7	3,3	3,5	3,2	3,4	3,0	3,6
5,1	5,7	5,3	5,5	5,2	5,4	5,0	5,6
2,1	2,7	2,3	2,5	2,2	2,4	2,0	2,6
4,1	4,7	4,3	4,5	4,2	4,4	4,0	4,6
0,1	0,7	0,3	0,5	0,2	0,4	0,0	0,6
6,1	6,7	6,3	6,5	6,2	6,4	6,0	6,6

Figure 5: The first two permutations can be collapsed to form the third. This is possible because the active cells in the first permutation correspond exactly to the inactive cells in the second and *vice versa*. Since the link contention in the first two permutations is 1, the combined permutation has link contention 2.

Table 2: Steps required as contention is allowed to increase.

Mesh size	Permitted Link Contention (c)							
($n \times n$)	1	2	3	4	5	6	7	8
4×4	16							
8×8	128	64						
12×12	432		144					
16×16	1024	512		256				
20×20	2000	1000			400			
24×24	3456	1728	1152			576		
28×28	5488						784	
32×32	8192	4096		2048				1024

consecutive sub-sequences of 2 or 4 permutations as shown in Figure 6. If the first synchronization in part (c) of this figure were removed we would have a schedule with node as well as link contention. Two nodes would be attempting to transmit at a time while the link contention would be doubled from 4 to 8. This can lead to further improvements in run time, as described below.

6 Implementation Considerations

The `nx` message passing library was used for our experiments on the Paragon. This library has its origins in the Intel iPSC-860 hypercube which has two types of messages: `FORCED` and `UNFORCED`. `FORCED` messages are transmitted from source to destination under the assumption that a receive has already been posted (i.e., buffer space for reception has been specified) at the destination. If an arriving message does not find a receive posted, it is discarded. `UNFORCED` messages do not require a receive to be posted beforehand. Before an `UNFORCED` message is transmitted there is an exchange of control messages between source and destination to allocate operating system buffer space for the message. This leads to additional overhead in communication (because of the control messages), extra memory requirements, and the penalty of copying from operating system buffers to user areas [3]. Further details of the communication overhead on the Paragon appear in [2]. Shirley et al. [10] discuss how operating system timer interrupts complicate performance measurement and prediction on this machine.

On the Paragon, `FORCED` and `UNFORCED` messages are *supposed* to perform identically. It has been our experience that operating system space is allocated for all possible arriving messages *in addition* to any user memory locations that may be set aside by explicitly posted receives. The user can specify the amount of memory buffers that the operating system is to set aside for this purpose. Despite this, when large numbers of large-sized messages are expected, the operating system can run

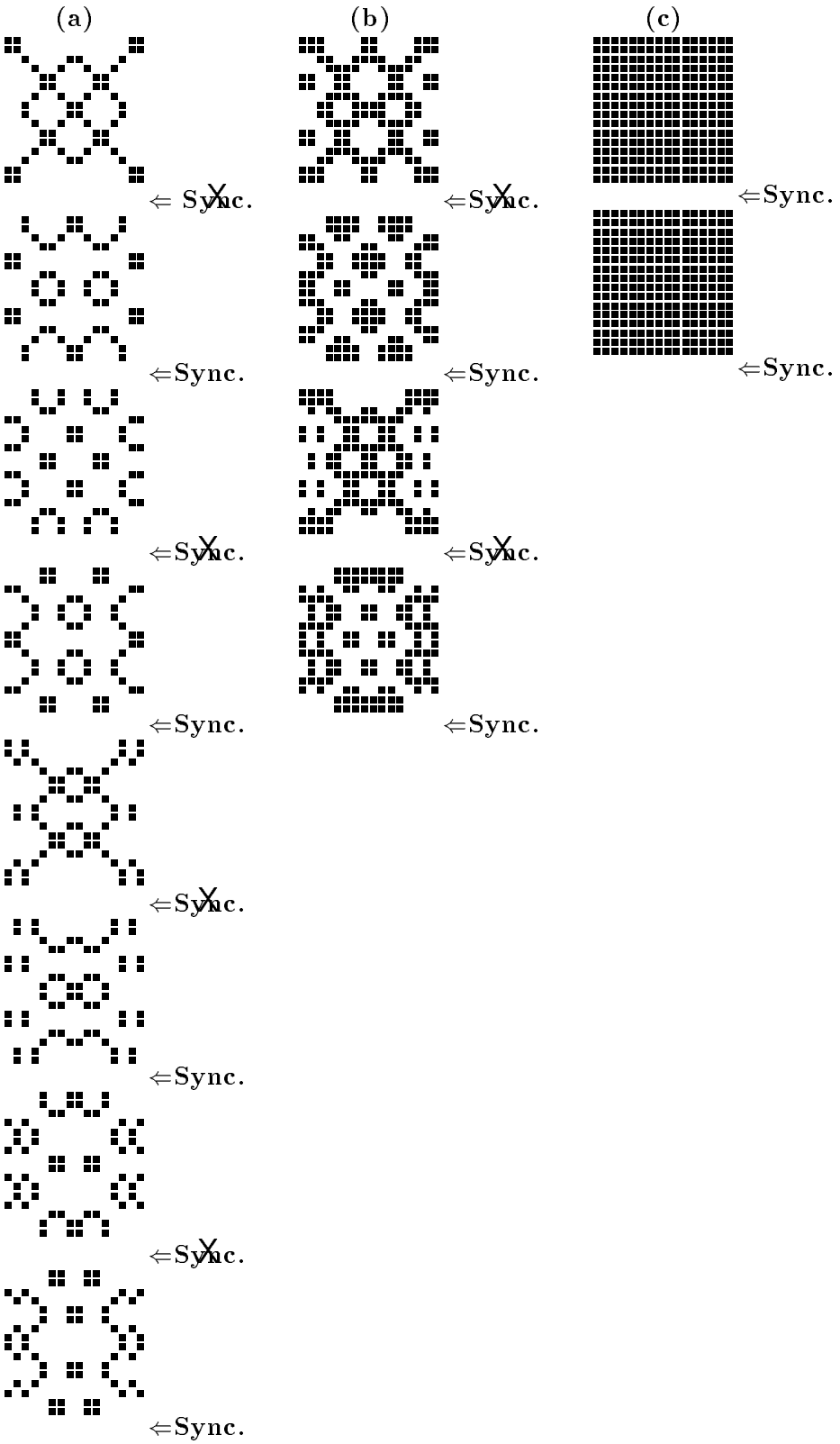

Figure 6: (a) The first 8 (of 1024) members of a collapsible schedule for a 16×16 mesh. Active nodes are indicated by square blocks. When alternate synchronizations are removed (X), pairs of successive permutations collapse as shown in part (b) giving a schedule with maximum link contention 2. Repeating this process results in a schedule with link contention 4 (c). Further removal of synchronization steps results in increasing node contention.

out of resources thereby causing the machine to hang. Needless to say, `FORCED` messages should only be used if communication requirements are well understood and receives can be posted before any messages are launched. Deadlocks can develop if this requirement is not satisfied.

The Bounded contention complete exchange algorithm that we have developed has a completely determined communication requirement and we could thus use `FORCED` messages. To compare the performance of the Bounded contention algorithm against an algorithm that does not take the topology of the mesh into account, we implemented a naive algorithm to carry out the complete exchange. This algorithm simply transmits blocks of data from each processor to the remaining processors without regard for link or node contention. We were unable to get the naive algorithm to function reliably beyond 12×12 processors because the large numbers of outstanding receives required could not be accommodated by the operating system.

Each node of the Paragon has an i860 processor dedicated to interprocessor communication. This processor takes over a considerable portion of the overhead of starting a data transfer. We have found that asynchronous receives and sends yield much better performance because the compute processor can spawn a task on the communication processor and carry on with its work without having to wait for the operation to complete. This, in fact, is how the machine manages to perform well under node contention.

Memory access and thus data communication on the Paragon is heavily affected by the starting address of a transfer. In our experiments we have aligned all arrays to 4k boundaries (the page size of the machine) to minimize this impact.

7 Experimental Results

When implementing Bounded contention complete exchange on the Paragon, several aspects of the machine performance had to be taken into account.

1. The amount of contention in a schedule can only be controlled by global synchronization. The overhead of this operation is substantial (Table 1).

2. While the machine can tolerate node and link contention, there is non-zero overhead associated with such contention.

3. Overheads for node and link contention are heavily dependent on the type of communication being carried out. It is very difficult to obtain simple expressions for these overheads. For example, measurements taken of the 1-dimensional communication patterns in Figure 2 do not apply to 2-dimensional communications.

The above aspects coupled with the use of virtual memory on the machine and the complex effects of operating system interrupts [10] make it extremely difficult to predict the communication performance of this machine under varying amounts of

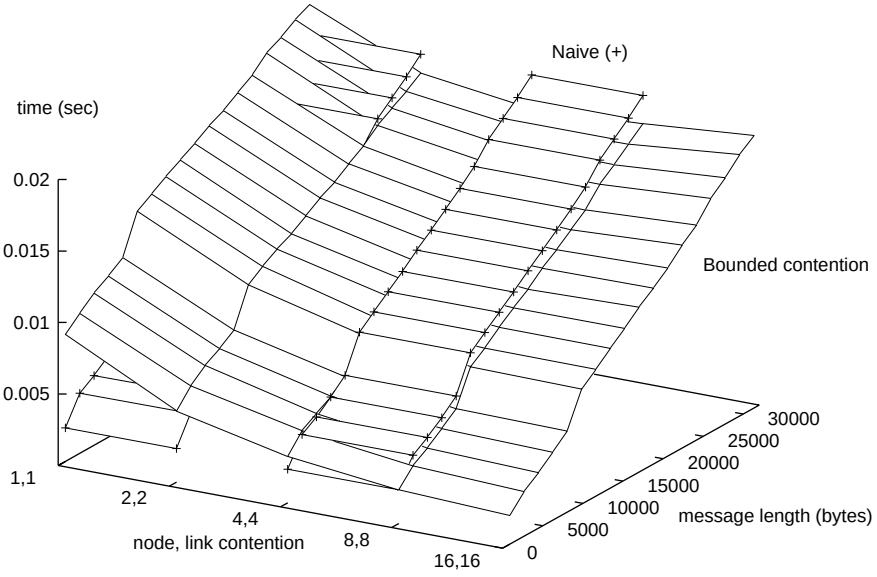

Figure 7: Naive algorithm ("+") compared with Bounded contention algorithm on a 4×4 Paragon. The naive algorithm run times, which do not vary with contention, have been shown as a series of strips for clarity.

node and link contention. This in turn also makes decision of the level of contention to be used difficult.

Our approach is to evaluate the algorithm for various levels of permitted contention and empirically decide on the best level for a given mesh size. This is easily done once a collapsible sequence has been generated for a mesh: simply insert barrier synchronizations in the sequence, modulo the permitted contention. Thus, for a 32×32 mesh we would insert barriers after every 1, 2, 4 or 8 permutations. For example, inserting barriers after every 4 permutations causes each group of 4 to collapse into one permutation with contention 4.

Figures 7, 8, 9 and 10 compare the performance of the naive and Bounded contention algorithms on meshes of size 4×4, 8×8, 12×12 and 16×16 respectively, for varying amounts of contention and message sizes. The x-axes of these plots are labeled with the pairs (node contention, link contention), as clarified in Figure 2. The performance of the naive algorithm, which does not vary with contention, is shown as a series of strips so that the surface of the Bounded algorithm can be seen clearly.

The small size of the 4×4 mesh does not permit a collapsible schedule to be generated (see Table 2). Despite this, there is an improvement in performance as contention increases, because the number of synchronization steps required is reduced. Furthermore, node contention also results in slight decreases in time as launching two or more messages in quick succession permits the utilization of intranode parallelism due to a separate communication processor.

Figures 8 and 9 show much more interesting results obtained from experiments on 8×8 and 12×12 meshes. Here, the performance of the Bounded algorithm

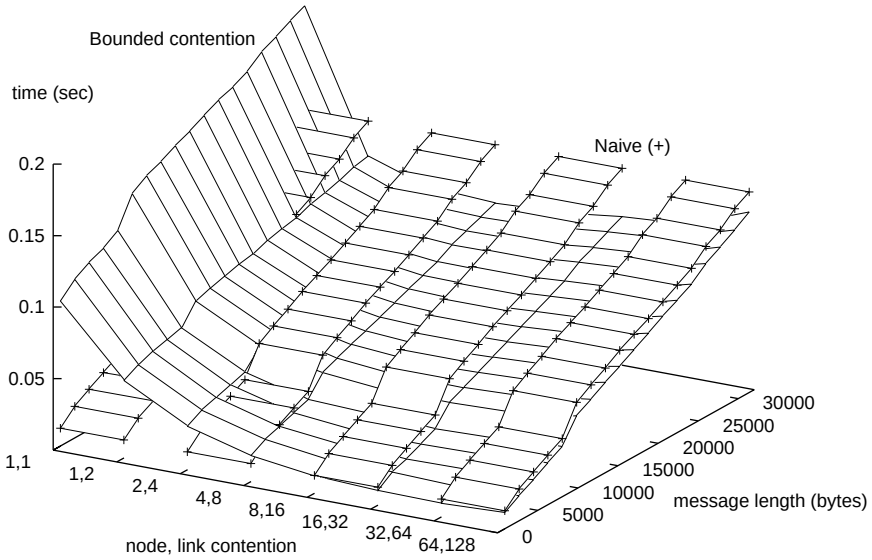

Figure 8: Comparison of the two algorithms on an 8×8 Paragon.

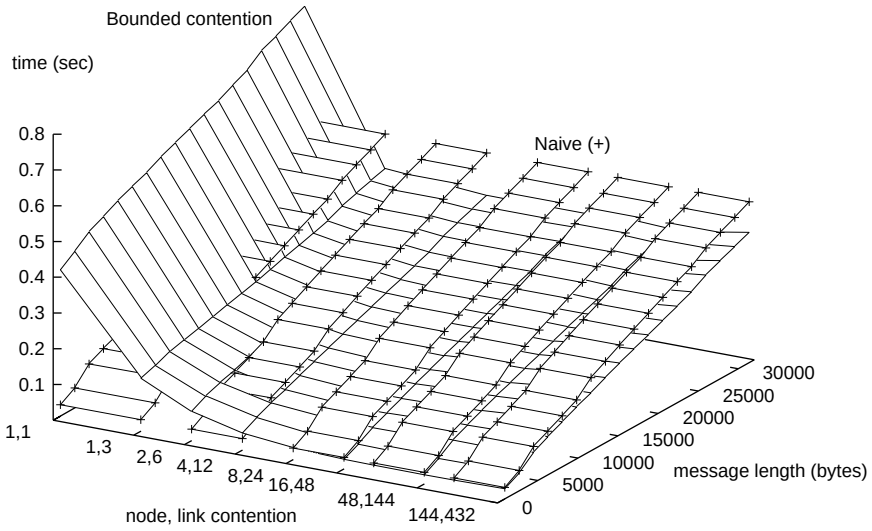

Figure 9: Comparison of the two algorithms on a 12×12 Paragon.

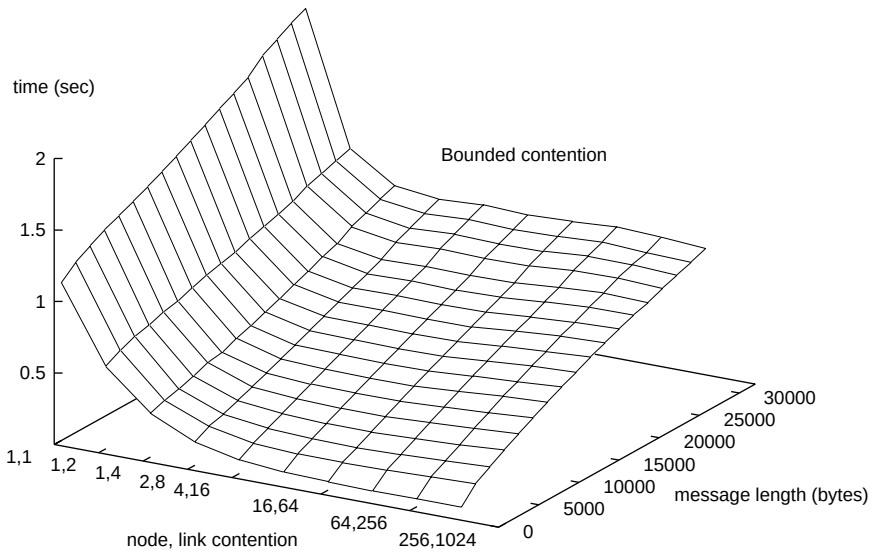

Figure 10: Performance of Bounded contention algorithm on a 16×16 Paragon. The naive algorithm fails to work on this mesh and the Bounded algorithm fails at contention $(256, 1024)$ because of operating system limitations.

is initially much poorer than the naive algorithm but improves very rapidly with increasing contention. The initial steep drop is due to the collapsing of the schedule, (which increases link contention but not node contention) and to the large reduction in synchronization steps. As contention increases, further improvements are obtained because of reduction in synchronization and because of the concurrent operation of the communication processor. However the improvement is arrested at node contention $= 16$ when the decrease in synchronization steps can no longer offset the overhead due to node and link contention. After this point the time starts increasing.

The performance of the Bounded algorithm for 16×16 meshes is shown in Figure 10. The Paragon failed to execute the naive algorithm for this mesh size. This is because the operating system could not allocate enough resources to accommodate the 256 receives required by the algorithm. The Bounded algorithm itself could not be be tested for this mesh size for node contention $= 256$ for the same reason.

The relative performance of the two algorithms is clear in Figure 11 which shows contours that indicate the percentage improvement of Bounded over naive. These contours show that improvements of greater than 25% are possible on 8×8 and 12×12 meshes for most message sizes, provided the contention level is chosen carefully. The contours help us pick the best contention level for a given message size.

To study our experimental results in greater detail we provide slices, at message size 15232 bytes, through the surfaces of Figures 7, 8, 9 & 10.

The solid curves in these figures show the measured time to execute Bounded contention complete exchange. This measured time is compared with the predicted time, obtained by adding synchronization and communication time taken from Table 1.

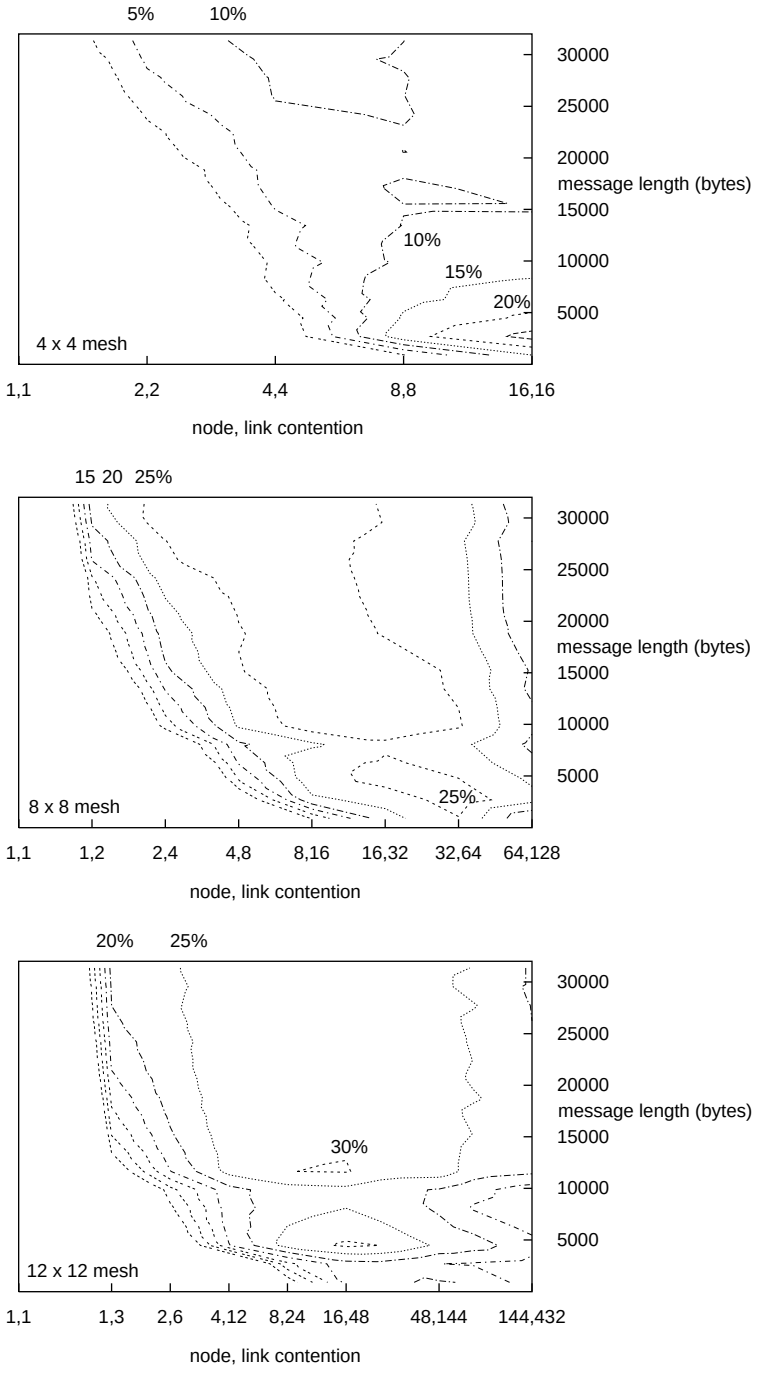

Figure 11: Contour plots of percentage improvement provided by Bounded contention algorithm over naive algorithm for 4×4, 8×8 and 12×12 Paragon meshes.

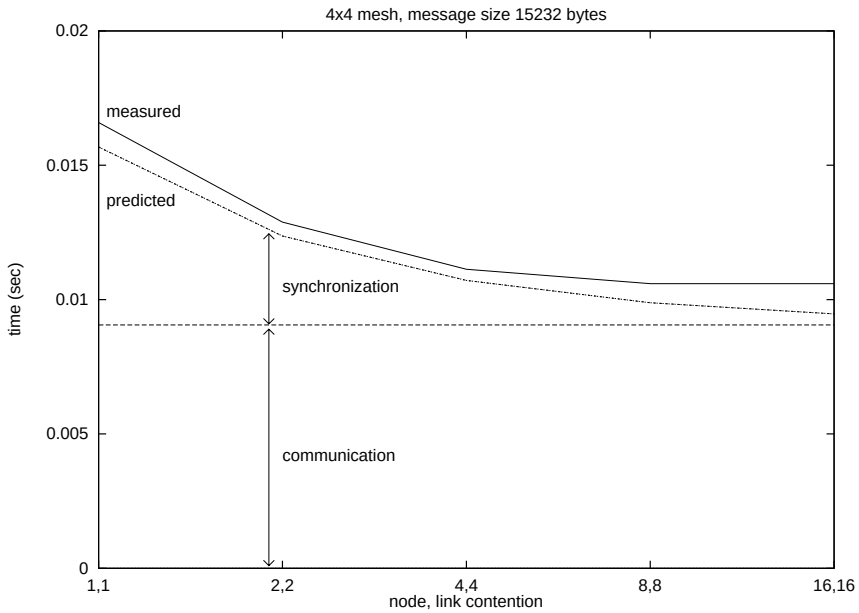

Figure 12: A slice through the surface for a 4×4 Paragon.

Figure 12 shows a slice through the surface for a 4×4 mesh (Figure 7). Three aspects of this figure are noteworthy.

- The agreement between predicted and measured times is good.
- The communication time fraction of the total predicted time is constant. This is because in a 4×4 mesh schedule there are no idle processors. Thus, even when we increase permitted contention, the schedule cannot collapse because of the lack of "holes" in the permutations.
- The increase in performance comes about because of reduction in synchronization overhead.

The slice of the 8×8 surface (Figure 13) brings out several interesting issues. To circumvent the difficulty of predicting performance we have inserted upper and lower bounds for time to execute complete exchange in this and subsequent figures. The lower bound gives the sum of communication and synchronization times as given in Table 1. Note that the communication time is halved going from link contention 1 to 2. This is because, as shown in Table 2, the number of communication steps drops from 128 to 64 for an 8×8 mesh. Since the lower bound does not include the overheads of node and link contention, the measured time should not drop below this curve.

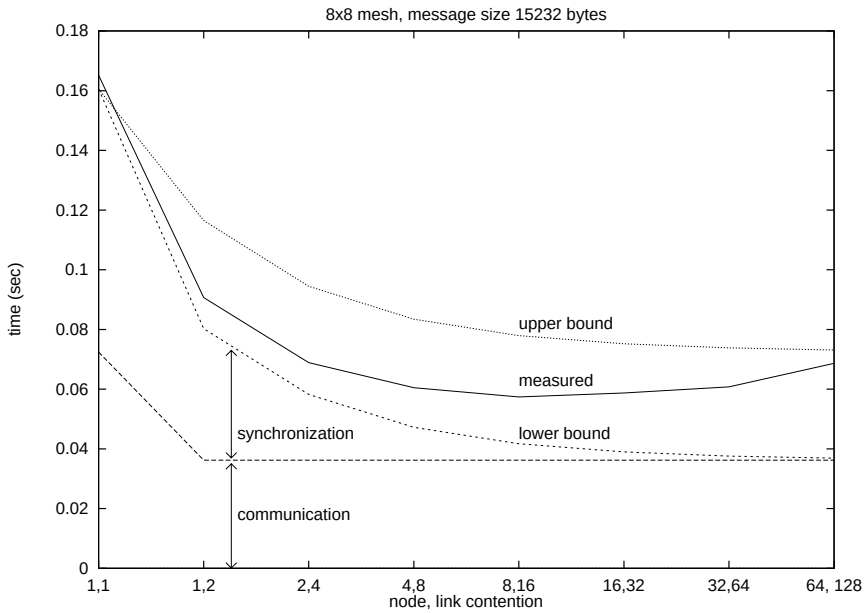

Figure 13: A slice through the surface for an 8×8 Paragon.

The Bounded contention algorithm increases permitted contention by deleting barriers. This removes control over the launching of messages: a processor can fire off the next message in its schedule without waiting for synchronization. Some messages may be launched along paths already in use, thereby increasing contention. The impact of this contention is very difficult to estimate because the communication patterns of the Bounded algorithm are complex and their contention cannot be characterized simply.

The upper bound curve gives the sum of synchronization and communication times, assuming that all 128 message steps are executed serially. We would expect the measured times to lie between the two bounds. The closer the measured time is to the lower bound, the greater is the success of the Bounded approach. On the other hand, the measured curve would approach the upper bound when the contention overheads exceed the reduction in communication and synchronization time.

In Figure 13 we see that the measured time is close to the lower bound for link contention 1, 2 & 4. Beyond 4 the measured time starts deviating significantly, reaching a minimum at link contention 16. Similar comments apply to the slices for 12×12 and 16×16 meshes (Figures 14 & 15). In the latter it is noteworthy that the measured time almost touches (but does not cross) the upper bound at contention (128,512). (Recall that this experiment could not be run for the last contention value of (256,1024) because of operating system limitations.) This shows that our algorithm is robust in the sense that the measured time remains bounded by the time to execute the individual communication steps.

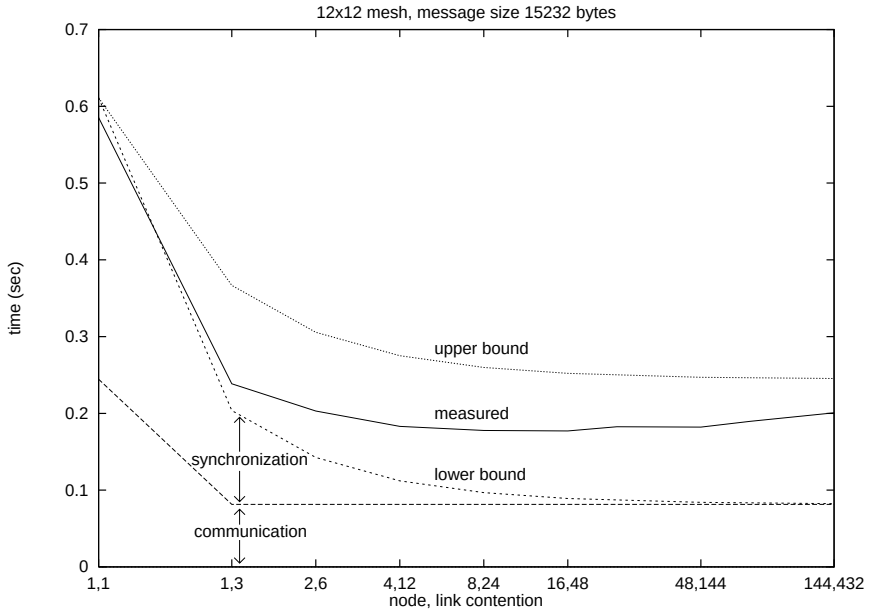

Figure 14: A slice through the surface for a 12×12 Paragon.

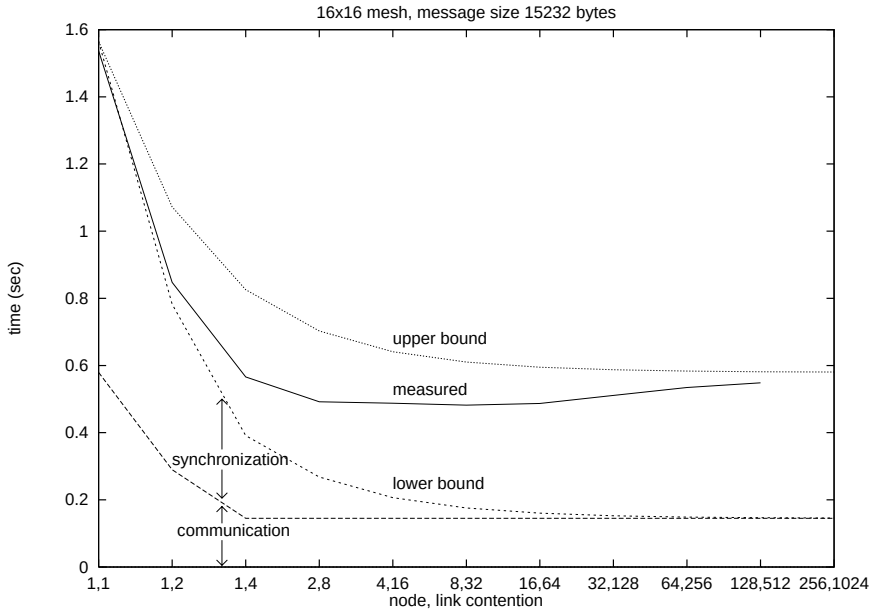

Figure 15: A slice through the surface for a 16×16 Paragon.

Figures 13, 14 and 15 show that a careful choice of contention levels is necessary to obtain the best performance. It is not enough to blindly remove all synchronization steps.

8 Conclusions

Complete exchange is an important communication requirement that is difficult to execute efficiently on meshes. We have developed a new Bounded contention algorithm that takes advantage of the high performance communication mechanism on the Paragon to achieve good timings. The performance of this algorithm has been measured to be better than that of a naive algorithm that does not take network topology into account. Our experience appears to contradict the commonly held belief that topology does not have to be considered when designing parallel algorithms for modern parallel computer systems.

Our results are applicable to all meshes in which, like the Paragon, the rate at which data can be transmitted *across* the interconnect is higher than the rate at which data can be injected *into* the interconnect. The successor to the Intel Paragon is the ASCI Teraflop machine with a *dual* mesh interconnect [12]. This machine can take advantage of our results in an interesting fashion. Our algorithm essentially "slices" the complete exchange communication pattern into a series of sub-patterns, each with a bounded contention. These sub-patterns can be alternately assigned to the two meshes permitting us to take full advantage of the ASCI's powerful interconnect. These results are also applicable to 3-d meshes because Scott's basic algorithm can be extended to higher dimensions.

An interesting area of further research would be to combine the Bounded algorithm which is optimal for large message sizes, with the multiphase algorithm [4] which has been shown to be applicable to the Paragon [5], and gives the best performance for small message sizes.

Perhaps the most crucial conclusion to be drawn from our experiments is the importance of synchronization time in determining the overall execution time of a communication step. Our results indicate that investment in an improved synchronization mechanism, perhaps relying on a network distinct from the network used for data communication, would yield handsome dividends in terms of improved communication performance.

Acknowledgments

We are grateful to Manuel Salas, Director ICASE, for his encouragement of this research. Discussions with Tom Crockett, Paul Fischer, David Keyes, Piyush Mehrotra, John Van Rosendale, Steve Seidel, and Xian-He Sun have been very valuable.

Access to the supercomputers at Caltech was arranged by Paul Messina. We wish to thank him and his able staff: Walker Aumann, Bevan Bennett, Sharon Burnett, Matthew Carle, Clark Chang, Shay Chinn, Alex Leung, Jan Lindheim, Heidi Lorenz-Wirzba, Julie Murphy, Mark L. Neidengard, Gary Dell'Osso, Andrew Sun, and Elsa Villate, for their alacrity in answering our queries and for their patient tolerance of the numerous crashes we caused on their machines. Thanh Phung, Al Bessey and Ellen Deleganes of Intel helped us with various problems on the Paragon.

References

[1] S. H. Bokhari and S. Berryman. Complete exchange on a circuit switched mesh. In *Proc. Scalable High Performance Computing Conf.*, pages 300–306, 1992.

[2] Shahid H. Bokhari. Communication overhead on the Intel Paragon, IBM SP2 and Meiko CS-2. ICASE Interim Report 28, NASA Contractor report 198211, September 1995. `http://www.icase.edu/docs/hilites/index.interim.html`.

[3] Shahid H. Bokhari. Communication overheads on the Intel iPSC-860 hypercube. ICASE Interim Report 10, May 1990.

[4] Shahid H. Bokhari. Multiphase complete exchange: A theoretical analysis. *IEEE Transactions on Computers*, 45(2):220–229, February 1996.

[5] Shahid H. Bokhari. Multiphase complete exchange on Paragon, SP2 and CS-2. *IEEE Parallel and Distributed Technology*, 3(4):45–59, Fall 1996.

[6] C-T. Ho and M. T. Raghunath. Efficient communication primitives on hypercubes. In *Proc. 6th. Conf. Distributed Memory Concurrent Computers*, pages 390–397, 1991.

[7] S. Lennart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, C-38(9):1249–1268, September 1989.

[8] T. Schmiermund and S. R. Seidel. A communication model for the Intel iPSC/2. Technical Report CS-TR 9002, Dept. of Computer Science, Michigan Tech. Univ., April 1990.

[9] D. S. Scott. Efficient all-to-all communication patterns in hypercube and mesh topologies. In *Proc. 6th. Conf. Distributed Memory Concurrent Computers*, pages 398–403, 1991.

[10] Hazel Shirley, Robert Reynolds, and Steve R. Seidel. Communication on the Intel Paragon. Technical Report CS-TR-95-07, Dept. of Computer Science, Michigan Tech. Univ., July 17, 1995.

[11] R. Take. A routing method for the all-to-all burst on hypercube network. In *Proc. 35th. National Conf. Info. Proc. Soc. Japan*, pages 151–152, 1987. In Japanese.

[12] Tom Thompson. The world's fastest computer (for now). *Byte*, 21(1):62, January, 1996.